A lambda-calculus with constructors

Ariel Arbiser, Alexandre Miquel, Alejandro Ríos

A λ-calculus with constructors

Ariel Arbiser[1], Alexandre Miquel[2], and Alejandro Ríos[1]

[1] Departamento de Computación – Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires, Argentina
{arbiser,rios}@dc.uba.ar
[2] PPS & Université Paris 7 – Case 7014, 2 Place Jussieu
75251 PARIS Cedex 05 – France.
alexandre.miquel@pps.jussieu.fr

Abstract. We present an extension of the $\lambda(\eta)$-calculus with a case construct that propagates through functions like a head linear substitution, and show that this construction permits to recover the expressiveness of ML-style pattern matching. We then prove that this system enjoys the Church-Rosser property using a semi-automatic 'divide and conquer' technique by which we determine all the pairs of commuting subsystems of the formalism (considering all the possible combinations of the nine primitive reduction rules). Finally, we prove a separation theorem similar to Böhm's theorem for the whole formalism.

1 Introduction

Lambda-calculus has been introduced by Church in the 30's [6] as a universal language to express computations of functions. Despite its remarkable simplicity, λ-calculus is rich enough to express all recursive functions. Since the rise of computers, λ-calculus has been used fruitfully as the basis of all functional programming languages, from LISP to the languages of the ML family. From the theoretical point of view, untyped λ-calculus enjoys many good properties [3], such as Church and Rosser's property expressing determinism of computations. In Logic, λ-calculus is also a fundamental tool to describe the computational contents of proofs via the Curry-Howard correspondence.

Although arbitrarily complex data structures can be encoded in the pure λ-calculus, modern functional programming languages provide primitive constructs for most data structures, for which a purely functional encoding would be inefficient. One of the most popular extensions of λ-calculus is pattern-matching on constructed values (a.k.a. variants), a problem that has been widely investigated in functional programming [12, 9, 13] and in rewriting [14, 7, 5, 11, 10].

However, introducing objects of different kinds—functions and constructed values—in the same formalism addresses the problem of their interaction. What does it mean to apply a constructed value to an argument? Should the constructed value accumulate the extra argument? Or should it produce an error? Similarly, what does it mean to perform case analysis on a function?

Unfortunately, these problems are usually not addressed in the literature because they are irrelevant in a typed setting—applications go with functions,

case analyses with variants. However, one should not forget that one of the reasons of the success of the λ-calculus in computer science and in logic lies in its excellent operational semantics in the untyped case. The best example is given by Böhm's separation theorem [4] that expresses that two observationally equivalent $\beta\eta$-normal λ-terms are intentionally equal. In the pure λ-calculus, $\beta\eta$-normal terms are not canonical forms because they cannot be further reduced; they are canonical forms because the computational behaviour of a $\beta\eta$-normal term cannot be expressed by another $\beta\eta$-normal term.

The situation is far from being as clear when we add pattern-matching to the untyped λ-calculus. As far as we know, there is no generalisation of Böhm's theorem for this kind of extension. One reason for that is that the notion of normal form is not as clear as in the pure λ-calculus, precisely because the traditional operational semantics says nothing about the computational behaviour of ill-typed constructions, such as a case analysis over an abstraction.

An extended operational semantics of case analysis In this paper, we propose an extension of the untyped λ-calculus with constructors and case analysis that fills the holes of the traditional operational semantics. Technically, the main novelty is that we let application and case analysis (written $\{\!|\theta|\!\}.\, M$) commute via the (ill-typed[1]) reduction rule

$$(\textsc{CaseApp}) \qquad \{\!|\theta|\!\}.\, (M\, N) \quad \to \quad (\{\!|\theta|\!\}.\, M)N\,.$$

(Here, θ denotes a *case binding*, that is a finite map from constructors to terms.) Symmetrically, we introduce a reduction rule

$$(\textsc{CaseLam}) \qquad \{\!|\theta|\!\}.\, (\lambda x\,.\, N) \quad \to \quad \lambda x\,.\, (\{\!|\theta|\!\}.\, M) \qquad (x \notin FV(\theta))$$

to let case analysis go through abstractions. In this way, case analysis can be understood as a form of head linear explicit substitution... of constructors.

Surprisingly, the system we obtain is not only computationally sound—we will show (section 3) that it is confluent and conservative over the untyped $\lambda\eta$-calculus—but it also permits to decompose ML-style pattern matching (with patterns of any arity) from the construction $\{\!|\theta|\!\}.\, M$ that only performs case analysis on constant constructors (section 2).

Finally, we will show (section 4) a theorem of weak separation for the whole calculus, using a separation technique inspired by Böhm's [4, 3]. For this reason, the formalism provides a special constant written ✠ and called the *daimon* (following the terminology and notation of [8]) that requests the termination of the program—something like an `exit` system call—and which will be used as the main technical device to observe normal forms and separate them.

Proofs and technical details are omitted from this extended abstract, but are available in the long version of the paper [1].

[1] Observe that M is treated as a function in the l.h.s. of the rule whereas it is treated as a constructed value in the r.h.s. This rule should not be confused with the rule of *commutative conversion* $(\{\!|\theta|\!\}.\, M)N = \{\!|\theta N|\!\}.\, M$ that comes from logic, a rule which is well-typed... but incompatible with the reduction rules of our calculus!

2 Syntax and reduction rules

2.1 Syntax

The λ-calculus with constructors distinguishes two kinds of names: *variables* (written x, y, z, etc.) and *constructors* (written c, c', etc.) The set of variables and the set of constructors are written $\mathcal{V}$ and $\mathcal{C}$, respectively. In what follows, we assume that both sets $\mathcal{V}$ and $\mathcal{C}$ are denumerable and disjoint.

The terms (written M, N, etc.) and the case bindings (written θ, ϕ, etc.) of the λ-calculus with constructors are inductively defined as follows:

Terms	$M, N ::= x$	(Variable)
	$\mid c$	(Constructor)
	$\mid \maltese$	(Daimon)
	$\mid MN$	(Application)
	$\mid \lambda x\,.\,M$	(Abstraction)
	$\mid \{\!\{\theta\}\!\}.\,M$	(Case construct)
Case bindings	$\theta, \phi ::= c_1 \mapsto M_1; \ldots; c_n \mapsto M_n$	$(c_i \neq c_j \text{ for } i \neq j)$

The sets of terms and case bindings are denoted by $\Lambda_{\mathcal{C}}$ and $\mathcal{B}$, respectively, and their disjoint union by $\Lambda_{\mathcal{C}}+\mathcal{B}$.

Constructor binding Each case binding θ is formed as a finite unordered list of constructor bindings of the form $(c \mapsto M)$ whose l.h.s. are pairwise distinct. We say that a constructor c is *bound* to a term M in a case binding θ if the binding $(c \mapsto M)$ belongs to the list θ. From the definition of case bindings, it is clear that a constructor c is bound to at most one term in a given case binding θ. When there is no such binding, we say that the constructor c is *unbound* in θ.

The *size* of a case binding $\theta = (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n)$ is written $|\theta|$ and defined by $|\theta| = n$.

We also introduce an (external) operation of *composition* between two case bindings θ and ϕ, which is written $\theta \circ \phi$ and defined by:

$$\theta \circ (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n) \quad \equiv \quad c_1 \mapsto \{\!\{\theta\}\!\}.\,M_1; \ldots; c_n \mapsto \{\!\{\theta\}\!\}.\,M_n$$

(where $\phi \equiv (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n)$). Notice that this operation is not syntactically associative, since:

$$(\theta \circ \phi) \circ (c_i \mapsto M_i)_{i=1..n} \quad \equiv \quad (c_i \mapsto \{\!\{\theta \circ \phi\}\!\}.\,M_i)_{i=1..n}$$

whereas

$$\theta \circ (\phi \circ (c_i \mapsto M_i)_{i=1..n} \quad \equiv \quad (c_i \mapsto \{\!\{\theta\}\!\}.\,\{\!\{\phi\}\!\}.\,M_i)_{i=1..n}$$

However, composition of case bindings only makes sense in the presence of the case conversion reduction rule $\{\!\{\theta\}\!\}.\,\{\!\{\phi\}\!\}.\,M \to \{\!\{\theta \circ \phi\}\!\}.\,M$ (see 2.2), for which both right hand sides above are convertible.

Free variables and substitution The notions of bound and free occurrences of a variable are defined as expected. The set of free variables of a term M (resp. a case binding θ) is written $FV(M)$ (resp. $FV(\theta)$).

As in the (ordinary) λ-calculus, terms are considered up to α-conversion (i.e. up to a renaming of bound variables). Notice that the renaming policy of the λ-calculus with constructors is strictly the same as in the λ-calculus: it only affects (bound) *variable names*, but leaves *constructor names* unchanged.

The external substitution operation of the λ-calculus, written $M\{x := N\}$, is extended to the λ-calculus with constructors as expected. The same operation is also defined for case bindings (notation: $\theta\{x := N\}$).

2.2 Reduction rules

The λ-calculus with constructors has 9 primitive reduction rules that are depicted in Fig. 1.

Beta-reduction			
AppLam	(AL)	$(\lambda x\,.\,M)N \to M\{x := N\}$	
AppDai	(AD)	$\maltese\, N \to \maltese$	
Eta-reduction			
LamApp	(LA)	$\lambda x\,.\,Mx \to M$	$(x \notin FV(M))$
LamDai	(LD)	$\lambda x\,.\maltese \to \maltese$	
Case propagation			
CaseCons	(CO)	$\{\!\{\theta\}\!\}.\, c \to M$	$((c \mapsto M) \in \theta)$
CaseDai	(CD)	$\{\!\{\theta\}\!\}.\maltese \to \maltese$	
CaseApp	(CA)	$\{\!\{\theta\}\!\}.\,(MN) \to (\{\!\{\theta\}\!\}.\,M)N$	
CaseLam	(CL)	$\{\!\{\theta\}\!\}.\,\lambda x\,.\,M \to \lambda x\,.\,\{\!\{\theta\}\!\}.\,M$	$(x \notin FV(\theta))$
Case conversion			
CaseCase	(CC)	$\{\!\{\theta\}\!\}.\,(\{\!\{\phi\}\!\}.\,M) \to \{\!\{\theta \circ \phi\}\!\}.\,M$	

Fig. 1. Reduction rules of the λ-calculus with constructors

In what follows, we will be interested not only in the system induced by the 9 reduction rules taken together, but more generally in the subsystems formed by all subsets of these 9 rules. We write $\lambda\mathcal{B}_\mathcal{C}$ the calculus generated by all rules of Fig. 1, and $\mathcal{B}_\mathcal{C}$ the calculus generated by all rules but AppLam (a.k.a. β).

Notice that AppLam (a.k.a β) and LamApp (a.k.a. η) are the only reduction rules that may apply to an ordinary λ-term in $\lambda\mathcal{B}_\mathcal{C}$.

2.3 An example

In $\lambda\mathcal{B}_\mathcal{C}$, the predecessor function (over unary integers) is implemented as

$$\mathrm{pred} \equiv \lambda n\,.\,\{\!|\mathsf{0} \mapsto \mathsf{0};\ \mathsf{s} \mapsto \lambda z\,.\,z|\!\}.\,n$$

(where $\mathsf{0}$ and s are two distinct constructors). From the rules AppLam ($=\beta$) and CaseCons it is obvious that

$$\mathrm{pred}\ \mathsf{0}\ \to\ \{\!|\mathsf{0} \mapsto \mathsf{0};\ \mathsf{s} \mapsto \lambda z\,.\,z|\!\}.\,\mathsf{0}\ \to\ \mathsf{0}\,.$$

More interesting is the case of pred $(\mathsf{s}\ N)$ (where N is an arbitrary term)

$$\begin{aligned}\mathrm{pred}\ (\mathsf{s}\ N) &\to \{\!|\mathsf{0} \mapsto \mathsf{0};\ \mathsf{s} \mapsto \lambda z\,.\,z|\!\}.\,(\mathsf{s}\ N)\\ &\to (\{\!|\mathsf{0} \mapsto \mathsf{0};\ \mathsf{s} \mapsto \lambda z\,.\,z|\!\}.\,\mathsf{s})\ N\ \to\ (\lambda z\,.\,z)\ N\ \to\ N\end{aligned}$$

which shows how the case construct captures the head occurrence of the constructor s via the reduction rule CaseApp. More generally, ML-style pattern-matching (on disjoint patterns) is translated in $\lambda\mathcal{B}_\mathcal{C}$ as follows:

$$\begin{array}{l}\texttt{match}\ N\ \texttt{with}\\ \mid\ \ c_1(x_1,\ldots,x_{n_1})\ \mapsto\ M_1\\ \mid\ \ c_2(x_1,\ldots,x_{n_2})\ \mapsto\ M_2\\ \mid\ \ \cdots\end{array}\qquad\text{becomes}\qquad\begin{array}{l}\{\!|c_1 \mapsto \lambda x_1\cdots x_{n_1}\,.\,M_1\,;\\ \ \ c_2 \mapsto \lambda x_1\cdots x_{n_2}\,.\,M_2\,;\\ \ \ \cdots\\ |\!\}\cdot N\end{array}$$

3 The Church-Rosser property

In this section, we aim to prove that $\lambda\mathcal{B}_\mathcal{C}$ is confluent. For that, we will prove a much more general result by characterising among the $2^9 = 512$ possible subsets of the 9 primitive reduction rules which subsets induce a subsystem of $\lambda\mathcal{B}_\mathcal{C}$ which is confluent, and which ones do not.

3.1 Preliminary definitions

Let us first recall some classic definitions.

Definition 1. — *An* Abstract Rewriting System (ARS) *is a pair $A = (|A|, \to_A)$ formed by an arbitrary set $|A|$ (called the carrier of A) equipped with a binary relation $\to_A$ on $|A|$. We denote by $\to_A^*$ the reflexive-transitive closure of $\to_A$, and by $\to_A^=$ the reflexive closure of $\to_A$.*

Definition 2. — *An ARS A is strongly normalising (SN) if there is no infinite sequence of objects $(M_i)_{i\in\mathbb{N}} \in |A|^\mathbb{N}$ such that $M_i \to_A M_{i+1}$ for all $i \in \mathbb{N}$.*

Definition 3. — *Let $A = (S, \to_A)$ and $B = (S, \to_B)$ be two ARSs defined on the same carrier set S. We say that:*

- *A* weakly commutes *with B, written $A \,/\!/_w\, B$, if for all M, M_1, M_2 s.t. $M \to_A M_1$ and $M \to_B M_2$ there exists M_3 s.t. $M_1 \to_B^* M_3$ and $M_2 \to_A^* M_3$.*

– A commutes *with* B, *written* $A \mathbin{//} B$, *if for all* M, M_1, M_2 *s.t.* $M \to_A^* M_1$ *and* $M \to_B^* M_2$ *there exists* M_3 *s.t.* $M_1 \to_B^* M_3$ *and* $M_2 \to_A^* M_3$.

An ARS A *is said to be* weakly confluent *or* weakly Church-Rosser (WCR) *(resp.* confluent, *or* Church-Rosser (CR)*) if* $A \mathbin{//_w} A$ *(resp. if* $A \mathbin{//} A$*).*

Given two ARSs A and B defined on the same carrier set, we write $A + B$ the (set-theoretic) union of both relations. The confluence proof of $\lambda\mathcal{B}_\mathcal{C}$ relies on standard results of rewriting [2], and in particular in the following two lemmas:

Lemma 1. — *If* $A \mathbin{//_w} B$ *and* $A + B$ *is SN, then* $A \mathbin{//} B$.

Proof: Same proof-technique as for Newman's lemma [2]. □

Lemma 2. — *If* $A \mathbin{//} B$ *and* $A \mathbin{//} C$ *then* $A \mathbin{//} (B + C)$.

3.2 Critical pairs and closure conditions

Each of the 9 primitive reduction rules of $\lambda\mathcal{B}_\mathcal{C}$ describes the interaction between two syntactic constructs of the language, which is reflected by the name of the rule: AppLam for 'Application over a Lambda', etc. These reduction rules induce 13 different critical pairs, that are summarised in Fig. 2 and 3.

Critical pairs occur for all pairs of rules of the form FooBar/BarBaz. A quick examination of Fig. 2 and 3 reveals that each time we have to close such a critical pair, we need to use the third rule FooBaz when this rule exists. This occurs for the 6 critical pairs (2), (4), (5), (6), (7) and (8) of Fig. 2; in the other cases, the critical pair is closed by the only rules FooBar and BarBaz.

This remark naturally suggests the following definition:

Definition 4 (Closure conditions). — *We say that a subset* s *of the 9 rules given in Fig. 1 fulfils the* closure conditions *and write* $s \models \mathrm{CC}$ *if:*

(CC1)	AppLam	$\in s$	$\wedge$	LamDai	$\in s$	$\Rightarrow$ AppDai $\in s$
(CC2)	LamApp	$\in s$	$\wedge$	AppDai	$\in s$	$\Rightarrow$ LamDai $\in s$
(CC3)	CaseApp	$\in s$	$\wedge$	AppLam	$\in s$	$\Rightarrow$ CaseLam $\in s$
(CC4)	CaseApp	$\in s$	$\wedge$	AppDai	$\in s$	$\Rightarrow$ CaseDai $\in s$
(CC5)	CaseLam	$\in s$	$\wedge$	LamApp	$\in s$	$\Rightarrow$ CaseApp $\in s$
(CC6)	CaseLam	$\in s$	$\wedge$	LamDai	$\in s$	$\Rightarrow$ CaseDai $\in s$

Intuitively, a subset that fulfils the 6 closure conditions defines a system in which all critical pairs can be closed, and thus constitutes a good candidate for Church-Rosser. The aim of this section is to turn this intuition into the

Theorem 1 (Church-Rosser). — *For each of the* 512 *subsystems* s *of* $\lambda\mathcal{B}_\mathcal{C}$ *the following propositions are equivalent:*

1. s *fulfils the closure conditions (CC1)–(CC6);*
2. s *is weakly confluent;*
3. s *is confluent.*

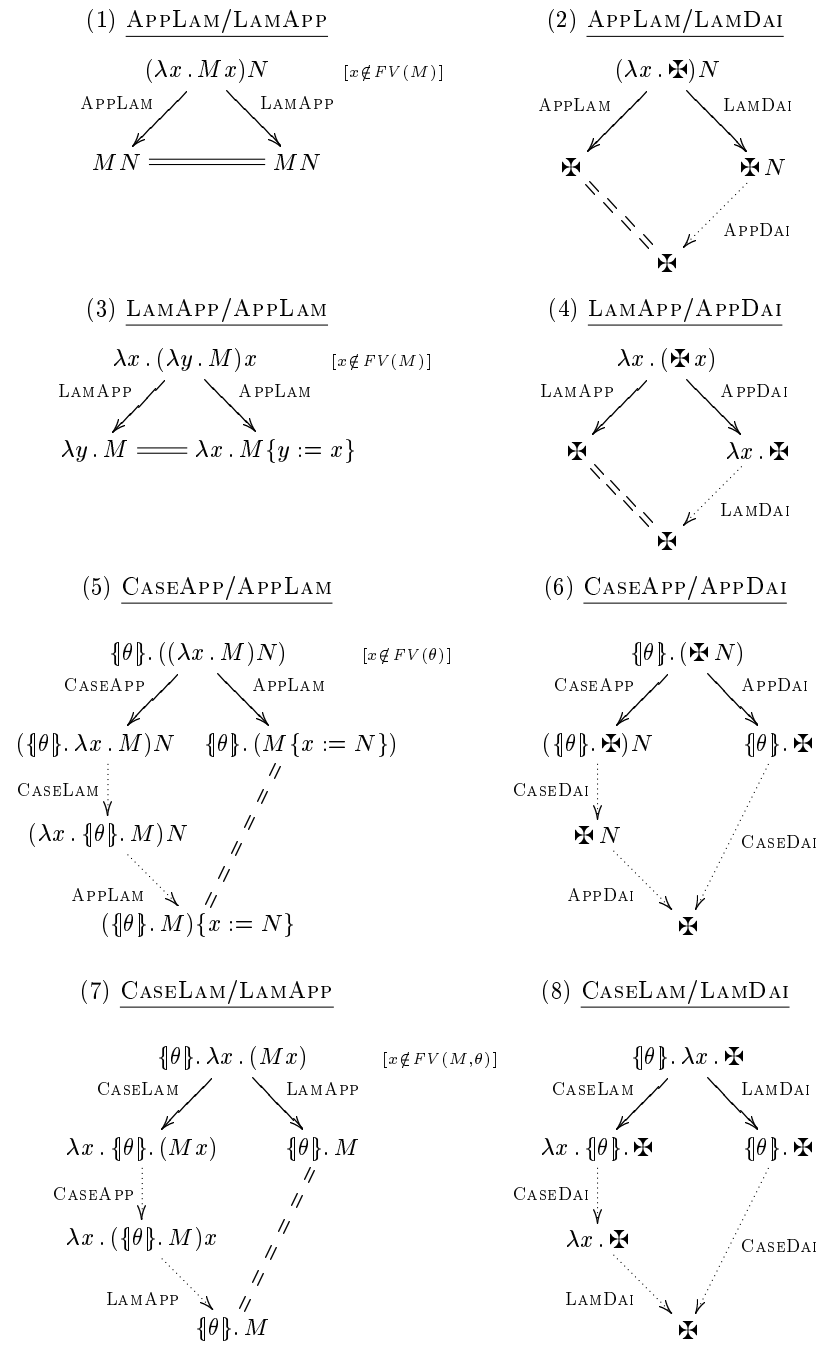


Fig. 2. Critical pairs 1–8 (/13)

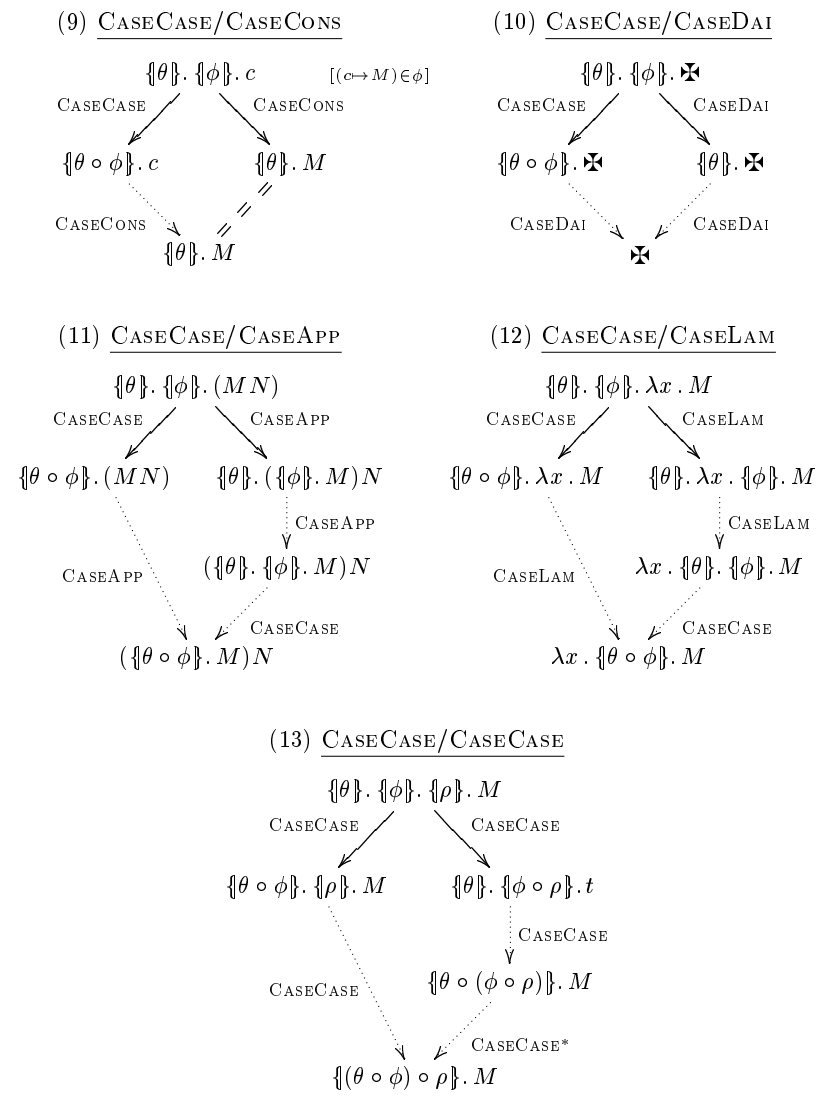


Fig. 3. Critical pairs 9–13 (/13)

Since the full system (i.e. $\lambda\mathcal{B}_\mathcal{C}$) obviously fulfils all closure conditions, we will get as an immediate corollary:

Corollary 1 (Church-Rosser). — *$\lambda\mathcal{B}_\mathcal{C}$ is confluent.*

The proof of theorem 1 relies on a systematic analysis of the commutation properties of all pairs of subsystems (s_1, s_2) of $\lambda\mathcal{B}_\mathcal{C}$. For that, we first have to generalise the notion of closure condition to any pair (s_1, s_2) of subsystems. This leads us to adopt the following definition:

Definition 5 (Binary closure conditions). — *We say that a pair (s_1, s_2) of subsystems fulfils the* binary closure conditions *and write $(s_1, s_2) \models \mathrm{BCC}$ if*

(BCC1)	AppLam	$\in s_1$	$\wedge$	LamDai	$\in s_2$	$\Rightarrow$	AppDai	$\in s_1$
(BCC2)	LamApp	$\in s_1$	$\wedge$	AppDai	$\in s_2$	$\Rightarrow$	LamDai	$\in s_1$
(BCC3)	CaseApp	$\in s_1$	$\wedge$	AppLam	$\in s_2$	$\Rightarrow$	CaseLam	$\in s_2$
(BCC4)	CaseApp	$\in s_1$	$\wedge$	AppDai	$\in s_2$	$\Rightarrow$	CaseDai	$\in (s_1 \cap s_2)$
(BCC5)	CaseLam	$\in s_1$	$\wedge$	LamApp	$\in s_2$	$\Rightarrow$	CaseApp	$\in s_2$
(BCC6)	CaseLam	$\in s_1$	$\wedge$	LamDai	$\in s_2$	$\Rightarrow$	CaseDai	$\in (s_1 \cap s_2)$
(BCC7)	CaseCase	$\in s_1$	$\wedge$	CaseDai	$\in s_2$	$\Rightarrow$	CaseDai	$\in s_1$
(BCC8)	CaseCase	$\in s_1$	$\wedge$	CaseApp	$\in s_2$	$\Rightarrow$	CaseApp	$\in s_1$
(BCC9)	CaseCase	$\in s_1$	$\wedge$	CaseLam	$\in s_2$	$\Rightarrow$	CaseLam	$\in s_1$

as well as the 9 symmetric conditions (obtained by exchanging s_1 with s_2).

Again, the 9 binary closure conditions come from an analysis of critical pairs. For example (BCC1) comes from the observation that critical pair (2) of Fig. 2 can be formed as soon as s_1 contains AppLam and s_2 contains LamDai, and that it can be closed only if s_1 contains AppDai.

We can also remark that when we take $s_1 = s_2 = s$, the binary closure conditions (BCC1)–(BCC6) degenerate to the (simple) closure conditions (CC1)–(CC6) whereas (BBC7)–(BCC9) become tautologies, so that:

Fact 1 — *For all subsystems s of $\lambda\mathcal{B}_\mathcal{C}$: $s \models \mathrm{CC}$ iff $(s, s) \models \mathrm{BCC}$.*

We first show that:

Proposition 1. — *For all pairs (s_1, s_2) of subsystems of $\lambda\mathcal{B}_\mathcal{C}$ the following propositions are equivalent:*

1. *$(s_1, s_2) \models \mathrm{BCC}$ (binary closure conditions);*
2. *$s_1 \,/\!/_w\, s_2$ (weak commutation).*

Proof: (1 $\Rightarrow$ 2) By structural induction on the reduced term, closing critical pairs using BCCs. (2 $\Rightarrow$ 1) By contraposition, exhibiting a suitable counter-example for each BCC that does not hold. □

Now it remains to be shown that all weakly commuting pairs commute.

3.3 Strong normalisation of the $\mathcal{B}_\mathcal{C}$-calculus

The first step is to check that the subsystem $\mathcal{B}_\mathcal{C} = (\lambda\mathcal{B}_\mathcal{C} \setminus \textsc{AppLam})$ is SN.

Proposition 2 (SN of $\mathcal{B}_\mathcal{C}$-calculus). — *The $\mathcal{B}_\mathcal{C}$-calculus is SN.*

Proof: Consider the function $h : \Lambda_\mathcal{C} + \mathcal{B} \to \mathbb{N}$ recursively defined by

$$\begin{array}{rclcrcl} h(x) & = & h(c) & = & h(\maltese) & = & 1 \\ h(\lambda x\,.\,M) & = & h(M) + 1 & & & & \\ h(MN) & = & h(M) + h(N) & & & & \end{array} \qquad \begin{array}{rcl} h(\{\!|\theta|\!\}.\,M) & = & h(\theta) + (|\theta| + 2)h(M) \\ & & \\ h((c_i \mapsto M_i)_{i=1..n}) & = & \sum_{i=1}^{n} h(M_i) \end{array}$$

It is routine to check that h decreases at each $\mathcal{B}_\mathcal{C}$-reduction step. □

From Lemma 1 and Prop. 1 we get:

Proposition 3. — *If* $(s_1, s_2) \models \mathrm{BCC}$ *and* $\textsc{AppLam} \notin (s_1 + s_2)$, *then* $s_1 \mathbin{//} s_2$.

3.4 Propagation of commutation lemmas

Let us now consider the 512×512 matrix formed by all $131,328$ (unordered) pairs of subsystems of $\lambda\mathcal{B}_\mathcal{C}$[2]. With the help of a small computer program[3], we easily check that $13,396$ of the $131,328$ pairs of systems fulfil BCCs—and thus weakly commute. Moreover, $5,612$ of these $13,396$ weakly commuting pairs do not involve AppLam—and thus we know that they commute.

The situation is summarised in the following table:

	Pairs (s_1, s_2)	$s_1 = s_2$
SN + commuting ($= \neg\textsc{AppLam} + \mathrm{BCC}$)	5,612	160
Weakly commuting ($= \mathrm{BCC}$)	13,396	248
Total	131,328	512

The problem is now to check that the $13,396 - 5,612 = 7,784$ remaining weakly commuting pairs commute too. For that, we notice that:

Fact 2 — *If the 12 pairs of subsystems of Table 1 commute, then all* $13,396$ *weakly commuting pairs of systems commute.*

Again, this fact can be mechanically checked by considering the set formed by all $5,612$ SN-commuting pairs extended with the 12 pairs of Table 1, and by checking that the closure of this set of $5,624$ pairs under Lemma 2 yields the set of all $13,396$ pairs that fulfil BCCs. To conclude, it suffices to prove:

Proposition 4. — *The 12 pairs of Table 1 commute.*

The details of the 12 commutation proofs can be found in [1].

From that we deduce that all pairs of subsystems that fulfil BCCs commute, and the proof of Theorem 1 is now complete.

[2] In what follows, we count (s_1, s_2) and (s_2, s_1) as a single pair of systems.

[3] This program can be downloaded from the web pages of the authors.

(1)	AppLam // AppLam
(2)	AppLam // AppDai
(3)	AppLam // LamApp
(4)	AppLam // CaseCons
(5)	AppLam // CaseDai
(6)	AppLam // CaseLam
(7)	AppLam // CaseCase
(8)	AppLam + AppDai // LamDai
(9)	AppLam + AppDai // LamApp + LamDai
(10)	AppLam + CaseLam // CaseApp
(11)	AppLam + CaseLam // LamApp + CaseApp
(12)	AppLam + AppDai + CaseDai + CaseLam // LamApp + LamDai + CaseDai + CaseApp

Table 1. The 12 initial commutation lemmas

Corollary 2. — *$\lambda\mathcal{B}_\mathcal{C}$ is conservative over $\lambda\eta$-calculus, in the sense that:*

$$\forall M_1, M_2 \in \Lambda \quad (\lambda\mathcal{B}_\mathcal{C} \models M_1 = M_2 \quad \Rightarrow \quad \lambda\eta \models M_1 = M_2)\,.$$

Proof: Follows from Cor. 1 using the concluding remark of subsection 2.2. □

4 Separation

The aim of this section is to establish the theorem of (weak) separation, expressing that observationally equivalent normal terms are syntactically equal. For that, we will show that for all normal terms[4] $M_1 \not\equiv M_2$ of $\lambda\mathcal{B}_\mathcal{C}$ there exists a context $C[]$ such that $C[M_1]$ converges whereas $C[M_2]$ diverges—or vice-versa—using notions of convergence and divergence that will be precised.

Separation [4] can be understood as some kind of completeness of the formalism. Intuitively, it expresses that the calculus provides sufficiently many reduction rules to identify observationally equivalent terms, or—which is the same dually—that it provides sufficiently many syntactic constructs (i.e. observers) to discriminate different normal forms.

4.1 Quasi-normal forms

Let us first analyse the shape of normal forms in the calculus.

Definition 6 (Head term). — *We call a* head term *(and write H, H_1, H', etc.) any term that has one of the following four forms:*

Head term $\qquad H \;::=\; x \;\mid\; c \;\mid\; \{\!|\theta|\!\}.\, x \;\mid\; \{\!|\theta|\!\}.\, c \quad (c \notin \mathrm{dom}(\theta))$

[4] Actually, we will prove our separation theorem only for *completely defined* normal terms (cf subsection 4.2).

When a head term H is of one of the first three forms (variable, constructor, case binding on a variable), we say that H is defined. *When H is of the last form (case binding on an unbound constructor), we say that H is* undefined.

Definition 7 (Quasi-head normal form). — *A term M is said to be in* quasi-head normal form (quasi-hnf) *if it has one of the following two forms*

$$\textbf{\textit{Quasi-hnf}} \qquad M \quad ::= \quad \Lightarrow\!\!\!\!\!\!\Lightarrow \quad | \quad \lambda x_1 \cdots x_n \,.\, H N_1 \cdots N_k \qquad (n, k \geq 0)$$

where H is an arbitrary head term, called the head *of M, and where $N_1, \ldots, N_k$ are arbitrary terms.*

Here, the prefix 'quasi-' expresses that such terms are in head normal form w.r.t. all reduction rules, but (possibly) the rule LamApp ($=\eta$). In what follows, 'quasi-' systematically refers to 'all reduction rules but LamApp'.

As for head terms, we distinguish *defined* quasi-hnfs from *undefined* ones. We say that a quasi-hnf M is *defined* when either $M \equiv \lambda x_1 \cdots x_n \,.\, H N_1 \cdots N_k$ with H defined, or when $M \equiv \Lightarrow\!\!\!\!\!\!\Lightarrow$; and we say that M is *undefined* when $M \equiv \lambda x_1 \cdots x_n \,.\, (\{\!|\theta|\!\}.\, c) N_1 \cdots N_k$ with $c \notin \mathrm{dom}(\theta)$.

More generally, we call a *defined* term (resp. an *undefined* term) any term that reduces to a defined (resp. undefined) quasi-hnf. The class of defined terms is closed under arbitrary reduction, as for the class of undefined terms. Moreover, the class of undefined terms is closed under arbitrary substitution.

Definition 8 (Quasi-normal form). — *A term (resp. a case binding) is said to be in* quasi-normal form *when it is in normal form w.r.t. all the reduction rules but* LamApp *($=\eta$).*

Terms (resp. case bindings) that are in quasi-normal form are simply called *quasi-normal terms* (resp. *quasi-normal case bindings*). In particular, we call a *quasi-normal head term* any head term H which is in quasi-normal form. These notions have the following syntactic characterisation:

Proposition 5. — *Quasi-normal terms, quasi-normal head terms, and quasi-normal case bindings are (mutually) characterised by the following BNF:*

$$\begin{array}{lrcl}
\textbf{\textit{Q.n.-terms}} & N & ::= & \Lightarrow\!\!\!\!\!\!\Lightarrow \;\mid\; \lambda x_1 \cdots x_n \,.\, H N_1 \cdots N_k \\
\textbf{\textit{Q.n.-head-terms}} & H & ::= & x \;\mid\; c \;\mid\; \{\!|\theta|\!\}.\, x \;\mid\; \{\!|\theta|\!\}.\, c \quad (c \notin \mathrm{dom}(\theta)) \\
\textbf{\textit{Q.n.-case bind.}} & \theta & ::= & c_1 \mapsto N_1; \ldots; c_p \mapsto N_p
\end{array}$$

4.2 Separation contexts

The notion of *context with one hole* is defined in $\lambda\mathcal{B}_C$ as expected. The term obtained by filling the hole of a context $C[]$ with a term M is written $C[M]$, and the composition of two contexts $C[]$ and $C'[]$ is written $C'[C[]]$. In what follows, we will use contexts of a particular form, namely, *evaluation contexts*:

$$\textbf{Evaluation contexts} \qquad E[] \quad ::= \quad []N_1 \cdots N_n \quad | \quad (\{\!|\theta|\!\}.\, [])N_1 \cdots N_n$$

Notice that the composition $E'[E[]]$ of two evaluation contexts $E[]$ and $E'[]$ is not always an evaluation context, but that it always reduces to an evaluation context using zero, one or several steps of the CaseApp rule, possibly followed by a single step of the CaseCase rule.

The daimon ✠ which represents immediate termination naturally absorbs all evaluation contexts:

Lemma 3. — *In any evaluation context $E[]$ one has* $E[✠] \to^* ✠$.

Symmetrically, each sub-term of the form $\{\!|\theta|\!\}.\, c$ (with $c \notin \mathrm{dom}(\theta)$) blocks the computation process at head position so that undefined terms "absorb" all evaluation contexts as well:

Lemma 4. — *Given an undefined term U, the term $E[U]$ is undefined for all evaluation contexts $E[]$.*

The daimon ✠ and undefined terms are thus natural candidates to define the notion of separability:

Definition 9 (Separability). — *We say that two terms M_1 and M_2 are:*

- weakly separable *if there exists a context with one hole $C[]$ such that either:*
  - $C[M_1] \to^* ✠$ *and $C[M_2]$ is undefined, or*
  - $C[M_2] \to^* ✠$ *and $C[M_1]$ is undefined;*
- strongly separable *if there exists two contexts $C_1[]$ and $C_2[]$ such that*
  - $C_1[M_1] \to^* ✠$ *and $C_1[M_2]$ is undefined, and*
  - $C_2[M_2] \to^* ✠$ *and $C_2[M_1]$ is undefined.*

Since undefined terms cannot be separated from each other (because undefined heads block all computations), we have to exclude them[5] from our study:

Definition 10 (Completely defined quasi-normal term). — *A term M in quasi-normal form is said to be* completely defined *if it contains no sub-term of the form $\{\!|\theta|\!\}.\, c$, where $c \notin \mathrm{dom}(\theta)$.*

4.3 Disagreement

The separation theorem is proved in two steps:

1. First we define a syntactic relation between terms, called *disagreement at depth $d \in \mathbb{N}$*, and we show that any pair of distinct normal forms have η-expansions that disagree at some depth (this subsection).
2. Then we show (by induction on the depth of disagreement) that any pair of disagreeing quasi-normal terms are weakly separable (subsection 4.5).

Definition 11 (Skeleton equivalence). — *We say that two defined head terms H_1 and H_2 have* the same skeleton *and write $H_1 \approx H_2$ if either:*

[5] Semantically, this means that we identify undefined terms with non weakly normalisable terms, and thus interpret them as Ω (Scott's bottom).

- $H_1 \equiv H_2 \equiv x$ *for some variable* x*; or*
- $H_1 \equiv H_2 \equiv c$ *for some constructor* c*; or*
- $H_1 \equiv \{\!|\theta_1|\!\}.\, x$ *and* $H_2 \equiv \{\!|\theta_2|\!\}.\, x$ *for some variable* x *and for some* θ_1, θ_2 *such that* $\mathrm{dom}(\theta_1) = \mathrm{dom}(\theta_2)$.

Definition 12 (Disagreement at depth d). — *For each* $d \in \mathbb{N}$*, we define a binary relation on the class of completely defined quasi-normal terms, called the* disagreement relation at depth d*. This relation, written* $\mathrm{dis}_d(M_1, M_2)$ *('*M_1 *and* M_2 *disagree at depth* d*'), is defined by induction on* $d \in \mathbb{N}$ *as follows:*

- *(Base case) We write* $\mathrm{dis}_0(M_1, M_2)$ *if either:*
  - $M_1 = \Phi$ *and* $M_2 = \lambda x_1 \cdots x_n \,.\, H N_1 \cdots N_k$*; or*
  - $M_1 = \lambda x_1 \cdots x_n \,.\, H N_1 \cdots N_k$ *and* $M_2 = \Phi$*; or*
  - $M_1 = \lambda x_1 \cdots x_n \,.\, H_1 N_{1,1} \cdots N_{1,k_1}$ *and* $M_2 = \lambda x_1 \cdots x_n \,.\, H_2 N_{2,1} \cdots N_{2,k_2}$ *and* $H_1 \not\approx H_2$.
- *(Inductive case) For all* $d \in \mathbb{N}$*, we write* $\mathrm{dis}_{d+1}(M_1, M_2)$ *if* $M_1 = \lambda x_1 \cdots x_n \,.\, H_1 N_{1,1} \cdots N_{1,k_1}$ *and* $M_2 = \lambda x_1 \cdots x_n \,.\, H_2 N_{2,1} \cdots N_{2,k_2}$ *and* $H_1 \approx H_2$*, and if either*
  - $H_1 = \{\!|\theta_1|\!\}.\, y$ *and* $H_2 = \{\!|\theta_2|\!\}.\, y$ *for some case bindings* θ_1, θ_2 *and for some variable* y*, and there is a constructor* $c \in \mathrm{dom}(\theta_1) = \mathrm{dom}(\theta_2)$ *such that* $\mathrm{dis}_d(\theta_1(c), \theta_2(c))$*; or*
  - *There is a position* $1 \leq k \leq \min(k_1, k_2)$ *such that* $\mathrm{dis}_d(N_{1,k}, N_{2,k})$.

Lemma 5 (Cooking lemma). — *If* M_1 *and* M_2 *are completely defined normal terms (w.r.t. all reduction rules including* LamApp $= \eta$*) such that* $M_1 \not\equiv M_2$*, then one can find two completely defined quasi-normal terms* M_1' *and* M_2' *such that* $M_1' \to_\eta^* M_1$*,* $M_2' \to_\eta^* M_2$*, and* $\mathrm{dis}_d(M_1', M_2')$ *for some* $d \in \mathbb{N}$.

4.4 Ingredients for separation

Separating disagreeing quasi-normal terms relies on definitions and techniques that are fully described in [1]. Here we briefly present some of them.

Tuples In order to retrieve arbitrary sub-terms of a given normal form (the so called 'Böhm-out' technique), we need tuples that are encoded as in the pure λ-calculus as $\langle M_1; \ldots; M_n \rangle \equiv \lambda e \,.\, e M_1 \cdots M_n$. In what follows, we use a more general notation to represent partial application of the n-uple constructor to its first k arguments and waiting the remaining $n - k$ arguments:

$$\langle M_1; \ldots; M_k; *_{n-k} \rangle \;\equiv\; \lambda x_{k+1} \cdots x_n e \,.\, e M_1 \cdots M_k x_{k+1} \cdots x_n \qquad (0 \leq k \leq n)$$

With these notations, the n-uple constructor is written $\langle *_n \rangle$.

Encoding names Separation of distinct free variables is achieved by substituting them by easily separable closed terms. For that, we associate to each variable name x a unique Church numeral written $\mathtt{x}$ (using the same name written in typewriter face), which we call the *symbol* of x.

Substitutions A *substitution* is a finite association list which maps pairwise distinct variables to terms. A substitution σ can be applied to a term M, and the result (which is defined as expected) is written $M[\sigma]$.

Separation is achieved (Prop. 6) using a particular substitution σ_X^K parameterised by an integer $K \geq 0$ and a finite set of variables X, namely, the substitution that maps each variable $x \in X$ to the term $\langle \mathtt{x}; *_K \rangle$ representing the partial application of the $(K+1)$-uple constructor to the symbol of x.

4.5 The separation theorem

Let M be a term in quasi-normal form. We call the *application strength* of M the largest integer $k \geq 0$ such that M has a sub-term of the form $HN_1 \cdots N_k$.

Proposition 6 (Separation of disagreeing terms). — *Let $K \geq 0$ be a natural number, and M_1 and M_2 two completely defined quasi-normal terms whose application strength is less than or equal to K and such that M_1 and M_2 disagree at some depth $d \in \mathbb{N}$. Then there exists an evaluation context $E[]$ such that either*

- $E\big[M_1[\sigma_X^K]\big] \to^* \maltese$ *and* $E\big[M_2[\sigma_X^K]\big]$ *is undefined, or*
- $E\big[M_2[\sigma_X^K]\big] \to^* \maltese$ *and* $E\big[M_1[\sigma_X^K]\big]$ *is undefined;*

where X is any finite set of variables that contains at least the free variables of M_1 and M_2, and where σ_X^K is the substitution defined in subsection 4.4.

From this proposition and lemma 5 we easily conclude:

Theorem 2 (Separation). — *Let M_1 and M_2 be completely defined terms in normal form. If $M_1 \not\equiv M_2$, then M_1 and M_2 are weakly separable.*

5 Conclusion

We have introduced an extension of λ-calculus, $\lambda\mathcal{B}_\mathcal{C}$, in which pattern matching is implemented via a mechanism of case analysis that behaves like a head linear substitution over constructors. We have shown that the reduction relation of $\lambda\mathcal{B}_\mathcal{C}$ is confluent and conservative over the $\lambda\eta$-calculus, but also that it is complete in the sense that it provides sufficiently many reduction rules to identify all observationally equivalent normalisable terms.

Using the divide-and-conquer method for other proofs of confluence An original aspect of this work is the way we proved confluence by systematically studying the commutation properties of all pairs of subsystems of $\lambda\mathcal{B}_\mathcal{C}$. Surprisingly, the mechanical propagation rule "if A // B and A // C then A // $(B+C)$" (combined with the primitive knowledge of all commutation properties between subsystems that do not involve AppLam) is sufficient to reduce the proof of the expected 7,784 non-trivial commutation lemmas to only 12 primitive lemmas, that are established by hand. It would be interesting to investigate further to see whether the same method can be used to prove the confluence of other rewrite systems with many reduction rules—typically, systems with explicit substitutions.

A notion of Böhm tree for $\lambda\mathcal{B}_\mathcal{C}$ The separation theorem we proved suggests that head normal forms of $\lambda\mathcal{B}_\mathcal{C}$ could be the adequate brick to define a notion of Böhm-tree [4, 3] for $\lambda\mathcal{B}_\mathcal{C}$—and more generally, for ML-style pattern-matching. However, the fact that it is a *weak* separation theorem also suggests that the observational ordering is non-trivial on the set of normal forms. Characterising observational ordering on normal forms could be the next step to deepen our understanding of both operational and denotational semantics of $\lambda\mathcal{B}_\mathcal{C}$.

Which type system for $\lambda\mathcal{B}_\mathcal{C}$? The reduction rules CaseApp and CaseLam which are the starting point of this work deeply challenge the traditional intuition of the notion of type, for which functions and constructed values live in different worlds. However, the good operational semantics of the calculus naturally raises the exciting question of finding a suitable type system for $\lambda\mathcal{B}_\mathcal{C}$.

References

1. A. Arbiser, A. Miquel, and A. Ríos. A λ-calculus with constructors. Manuscript, available from the web pages of the authors, 2006.
2. F. Baader and T. Nipkow. *Rewriting and All That.* Addison-Wesley, 1999.
3. H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and The Foundations of Mathematics.* North-Holland, 1984.
4. C. Böhm, M. Dezani-Ciancaglini, P. Peretti, and S. Ronchi Della Rocha. A discrimination algorithm inside lambda-beta-calculus. *Theoretical Computer Science*, 8(3):265–291, 1979.
5. S. Cerrito and D. Kesner. Pattern matching as cut elimination. In *Logics In Computer Science (LICS'99)*, pages 98–108, 1999.
6. A. Church. *The calculi of lambda-conversion*, volume 6 of *Annals of Mathematical Studies.* Princeton, 1941.
7. H. Cirstea and C. Kirchner. Rho-calculus, the rewriting calculus. In *5th International Workshop on Constraints in Computational Logics*, 1998.
8. J.-Y. Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, 2001.
9. P. Hudak, S. Peyton-Jones, and P. Wadler. Report on the programming language Haskell, a non-strict, purely functional language (Version 1.2). Sigplan Notices, 1992.
10. C. Barry Jay. The pattern calculus. *ACM Transactions on Programming Languages and Systems*, 26(6):911–937, 2004.
11. W. Kahl. Basic pattern matching calculi: Syntax, reduction, confluence, and normalisation. Technical Report 16, Software Quality Research Laboratory, McMaster Univ., 2003.
12. R. Milner, M. Tofte, and R. Harper. *The definition of Standard ML.* MIT Press, 1990.
13. The Objective Caml language. `http://caml.inria.fr/`.
14. V. van Oostrom. Lambda calculus with patterns. Technical Report IR-228, Vrije Universiteit, Amsterdam, 1990.

A 12 initial commutation properties

In this appendix we show the 12 commutation properties of Table 1.

The first commutation property of Table 1 expresses the confluence of the reduction rule AppLam. As usual, we prove it (following Tait and Martin-Löf) by introducing the corresponding notion of parallel reduction:

Definition 13. — *The relations of* parallel AppLam-reduction *on terms and on case bindings (both written $\Rightarrow$) are defined as follows:*

$$\frac{}{M \Rightarrow M}\ (\textsc{pRef}) \qquad \frac{M \Rightarrow M' \qquad N \Rightarrow N'}{(\lambda x\,.\,M)N \Rightarrow M'\{x := N'\}}\ (\textsc{pAppLam})$$

$$\frac{M \Rightarrow M'}{\lambda x\,.\,M \Rightarrow \lambda x\,.\,M'}\ (\textsc{pLam}) \qquad \frac{M \Rightarrow M' \qquad N \Rightarrow N'}{MN \Rightarrow M'N'}\ (\textsc{pApp})$$

$$\frac{M \Rightarrow M' \qquad \theta \Rightarrow \theta'}{\{\!|\theta|\!\}.\,M \Rightarrow \{\!|\theta'|\!\}.\,M'}\ (\textsc{pCase})$$

$$\frac{M_1 \Rightarrow M_1' \quad \cdots \quad M_n \Rightarrow M_n'}{(c_i \mapsto M_i)_{i=1..n} \Rightarrow (c_i \mapsto M_i')_{i=1..n}}\ (\textsc{pCBind})$$

As usual, we check that:

Proposition 7 (Properties of $\Rightarrow$).

1. *If $M \to_\beta M'$, then $M \Rightarrow M'$ (i.e. $\to_\beta \subset \Rightarrow$)*
2. *If $M \Rightarrow M'$, then $M \to_\beta^* M'$ (i.e. $\Rightarrow \subset \to_\beta^*$)*
3. *If $M \Rightarrow M'$ and $N \Rightarrow N'$, then $M\{x := N\} \Rightarrow M'\{x := N'\}$*
4. *If $M \Rightarrow M_1$ and $M \Rightarrow M_2$, then there exists M_3 s.t. $M_1 \Rightarrow M_3$ and $M_2 \Rightarrow M_3$ (diamond property)*

Proof: Item 1: by induction on the derivation of $M \to_\beta M'$. Item 2: by induction on the derivation of $M \Rightarrow M'$. Item 3: by induction on the derivation of $M \Rightarrow M'$. Item 4: by induction on the derivation of $M \Rightarrow M_1$. □

From this we deduce that $\Rightarrow^* = \to_\beta^*$, and thus:

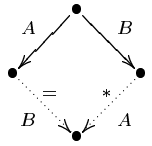
Proposition 8 (1/12). — AppLam // AppLam, *i.e.* AppLam *is confluent.*

The next 5 commutations properties (2–6) are of the form 'AppLam // r', where the reduction rule r is *linear*, that is, a rule that cannot duplicate sub-terms during contraction. (But it may erase sub-terms.)

To treat this case, we use the following definition:

Definition 14. — *Let $(S, \to_A)$ and $(S, \to_B)$ be two ARSs defined on the same carrier set. We say that A* strongly commutes *with B if for all M, M_1, M_2 such*

that $M \to_A M_1$ *and* $M \to_B M_2$ *there exists* M_3 *such that* $M_1 \to_{=B} M_3$ *and* $M_2 \to_A^* M_3$. *In other words, the following diagram holds:*



Lemma 6. — *If* A *strongly commutes with* B, *then* $A \,//\, B$.

PROOF: See [2]. □

Lemma 7. — *For each reduction rule*

$$r \in \{\text{AppDai}; \text{LamApp}; \text{CaseCons}; \text{CaseDai}; \text{CaseLam}\},$$

the rule r *strongly commutes with* AppLam.

PROOF: This is proved by a straightforward induction. Notice that the reduction rules AppDai, CaseCons, CaseDai and CaseLam induce no critical pair with AppLam. Only the rule LamApp induces critical pairs with AppLam, but these pairs are trivially closed. □

Proposition 9 (2–6/12). — *For each reduction rule*

$$r \in \{\text{AppDai}; \text{LamApp}; \text{CaseCons}; \text{CaseDai}; \text{CaseLam}\},$$

we have AppLam $//$ r.

The commutation between AppLam and CaseCase is more delicate to handle since both rules may duplicate redexes of the other kind during contraction. However, the problem is greatly simplified if we replace AppLam by $\Rightarrow$ (parallel AppLam-reduction), since:

Lemma 8. — CaseCase *strongly commutes with* $\Rightarrow$.

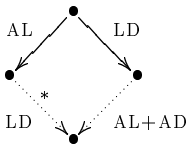
PROOF: By induction on the derivation of $\Rightarrow$. □

Proposition 10 (7/12). — AppLam $//$ CaseCase.

PROOF: By lemma 6, we know that CaseCase commutes with $\Rightarrow$. But since we know that $\Rightarrow^* = \to_\beta^*$, we are done. □

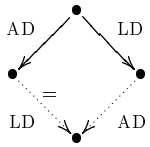
The following lemma describes the interaction between the reduction rules AppLam and LamDai, generalising critical pair (2) of Fig. 2:

Lemma 9 (AppLam/LamDai). — *The following diagram holds:*



PROOF: By structural induction on the initial term (top of the diagram). In all configurations where the initial AppLam- and LamDai-redexes are disjoint, contracting the AppLam-redex may duplicate the LamDai-redex (hence $\rightarrow_{\mathrm{LD}}^{*}$ to close on the lhs) whereas contracting the LamDai-redex leaves the AppLam-redex unaffected (hence $\rightarrow_{\mathrm{AL}}$ to close on the rhs). In the configuration of the critical pair AppLam/LamDai, we need no LamDai-reduction step to close on the lhs, but a single AppDai-reduction step to close on the rhs (hence the use of AD). □

Lemma 10 (AppDai/LamDai). — *The following diagram holds:*



PROOF: Obvious since both rules are linear and induce no critical pair. Notice that contracting the AppDai-redex may erase the LamDai-redex, hence the '=' to close on the lhs. □

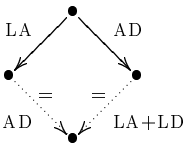
By merging the diagrams of lemmas 9 and 10 we deduce:

Lemma 11. — LamDai *strongly commutes with* AppLam + AppDai.

Proposition 11 (8/12). — AppLam + AppDai // LamDai.

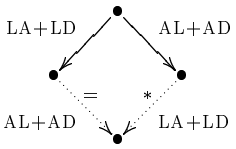
Again, the following lemma describes the interaction between the reduction rules LamApp and AppDai, generalising critical pair (4) of Fig. 2:

Lemma 12 (LamApp/AppDai). — *The following diagram holds:*

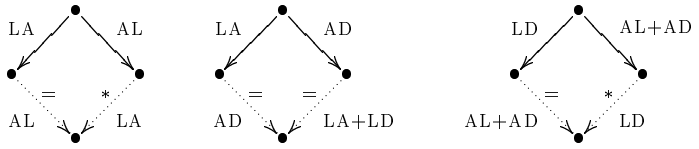


(The proof follows the same idea as for lemma 9.)

Lemma 13. — LamApp + LamDai *strongly commutes with* AppLam + AppDai*:*



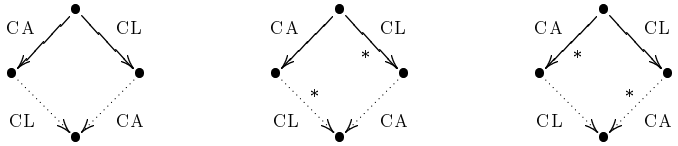
PROOF: The proof proceeds by merging the following diagrams, that cover all the possible cases when $M \to_{\text{LA+LD}} M_1$ and $M \to_{\text{AL+AD}} M_2$:



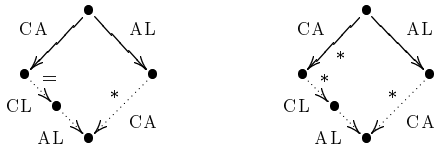
First diagram is lemma 7 with $r = \text{LAMAPP}$, second diagram is lemma 12, and third diagram is lemma 11. □

Proposition 12 (9/12). — APPLAM + APPDAI // LAMAPP + LAMDAI.

Lemma 14 (CaseApp/CaseLam). — *The following diagrams hold:*



Lemma 15 (CaseApp/AppLam). — *The following diagrams hold:*

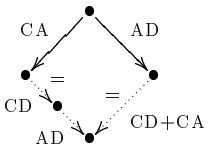


PROOF: The first diagram is obtained by generalising critical pair (5) of Fig. 2, following the spirit of lemmas 9 and 12. The second diagram is deduced from the first, by induction on the number of CASEAPP-reduction steps (top left), using the second diagram of lemma 14 to close. □

Proposition 13 (10/12). — APPLAM + CASELAM // CASEAPP.

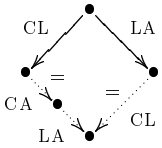
PROOF: By induction on the number of (APPLAM + CASELAM)-reduction steps, using the third diagram of lemma 14 and the second diagram of lemma 15. □

Lemma 16 (CaseApp/AppDai). — *The following diagram holds:*



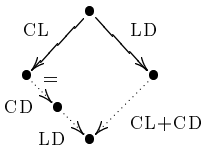
PROOF: This diagram is obtained by generalising critical pair (6) of Fig. 2, following the spirit of lemmas 9 and 12. □

Lemma 17 (CaseLam/LamApp). — *The following diagram holds:*



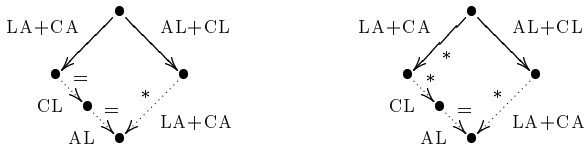
PROOF: This diagram is obtained by generalising critical pair (7) of Fig. 2. □

Lemma 18 (CaseLam/LamDai). — *The following diagram holds:*

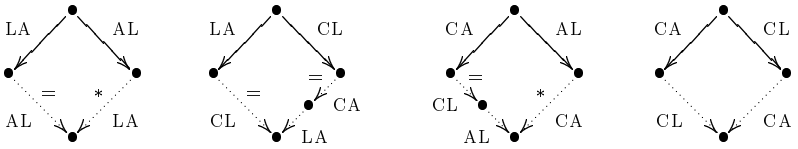


PROOF: This diagram is obtained by generalising critical pair (8) of Fig. 2. □

Lemma 19. — *The following diagrams hold:*



PROOF: The first diagram is obtained by merging the following diagrams that cover all the possible cases:

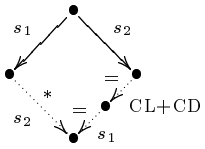


(The diagrams above come from lemmas 7, 17, 15 and 14, respectively.) The second diagram is deduced from the first diagram, by induction on the number of (LamApp + CaseApp)-reduction steps (see [1] for the details). □

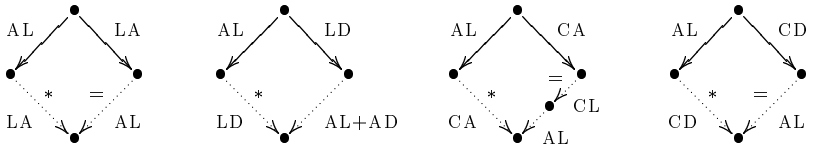
Proposition 14 (11/12). — AppLam + CaseLam // LamApp + CaseApp.

PROOF: From the second diagram of lemma 19, by induction on the number of (AppLam + CaseLam)-reduction steps. □

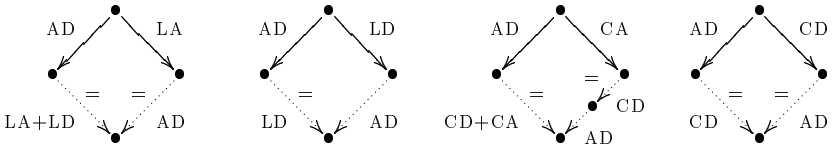
Lemma 20. — *Let* s_1 = AppLam + AppDai + CaseDai + CaseLam *and* s_2 = LamApp + LamDai + CaseDai + CaseApp. *The following diagram holds:*



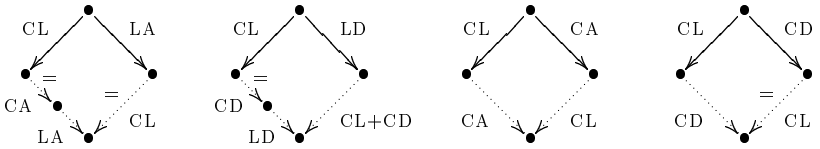
PROOF: The proof proceeds by merging the following diagrams, that cover all the possible cases:



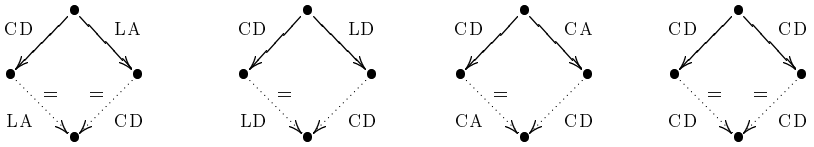
(Lemma 7) (Lemma 9) (Lemma 15) (Lemma 7)



(Lemma 12) (Lemma 10) (Lemma 16)



(Lemma 17) (Lemma 18) (Lemma 14)



(Non annotated diagrams describe the interaction between two linear rules that have no critical pair.) □

Proposition 15 (12/12). — AppLam + AppDai + CaseDai + CaseLam *commutes with* LamApp + LamDai + CaseDai + CaseApp.

PROOF: Again, let s_1 = AppLam + AppDai + CaseDai + CaseLam and s_2 = LamApp + LamDai + CaseDai + CaseApp. The proof of confluence is done by induction on the $(s_2 + \text{CL} + \text{CD})$-depth of the top term, using lemma 20 to close the diagram (see [1] for the details). □

B Confluence of the whole system $\lambda\mathcal{B}_\mathcal{C}$

Each item of the following (mechanically constructed) proof states a commutation property $(s_1 \,//\, s_2)$ which is either:

- an item of Table 1;
- a consequence of $(s_1, s_2) \models \mathrm{BCC}$ and $(s_1 + s_2) \models \mathrm{SN}$;
- a consequence of two former items using the rule of inference:

$$\text{if } A \,//\, B \text{ and } A \,//\, C, \text{ then } A \,//\, (B + C)\,.$$

1. (AL $\models$ CR) [Table 1:1]
2. (AL // AD) [Table 1:2]
3. (AL // CO) [Table 1:4]
4. (AL // CD) [Table 1:5]
5. (AL // CL) [Table 1:6]
6. (AL // CD + CL) since (CD // AL) [4.] and (CL // AL) [5.]
7. (AL // AD + CD + CL) since (AD // AL) [2.] and (CD + CL // AL) [6.]
8. (AL // AL + AD + CD + CL) since (AL $\models$ CR) [1.] and (AD + CD + CL // AL) [7.]
9. (AL // CC) [Table 1:7]
10. (LA + LD + CD + CA // AL + AD + CD + CL) [Table 1:12]
11. (AL // CL + CC) since (CL // AL) [5.] and (CC // AL) [9.]
12. (AL // CD + CL + CC) since (CD // AL) [4.] and (CL + CC // AL) [11.]
13. (AL // CO + CD + CL + CC) since (CO // AL) [3.] and
    (CD + CL + CC // AL) [12.]
14. (AL // AD + CO + CD + CL + CC) since (AD // AL) [2.] and
    (CO + CD + CL + CC // AL) [13.]
15. (AD + CD + CL // AD + CO + CD + CL + CC) since BCC + SN
16. (AL + AD + CD + CL // AD + CO + CD + CL + CC) since
    (AL // AD + CO + CD + CL + CC) [14.] and
    (AD + CD + CL // AD + CO + CD + CL + CC) [15.]
17. (AL + AD + CD + CL // AD + LA + LD + CO + CD + CA + CL + CC) since
    (LA + LD + CD + CA // AL + AD + CD + CL) [10.] and
    (AD + CO + CD + CL + CC // AL + AD + CD + CL) [16.]
18. (AL + AD + CD + CL // AL + AD + LA + LD + CO + CD + CA + CL + CC)
    since (AL // AL + AD + CD + CL) [8.] and
    (AD + LA + LD + CO + CD + CA + CL + CC // AL + AD + CD + CL) [17.]
19. (LA+LD+CO+CD+CA+CL+CC // AD+LA+LD+CO+CD+CA+CL+CC)
    since BCC + SN
20. (AD + LA + LD + CO + CD + CA + CL + CC //
    AL + AD + LA + LD + CO + CD + CA + CL + CC) since
    (AL + AD + CD + CL // AD + LA + LD + CO + CD + CA + CL + CC) [17.] and
    (LA + LD + CO + CD + CA + CL + CC //
    AD + LA + LD + CO + CD + CA + CL + CC) [19.]
21. (AL + AD + LA + LD + CO + CD + CA + CL + CC $\models$ CR) since
    (AL + AD + CD + CL // AL + AD + LA + LD + CO + CD + CA + CL + CC) [18.]
    and (AD + LA + LD + CO + CD + CA + CL + CC //
    AL + AD + LA + LD + CO + CD + CA + CL + CC) [20.]